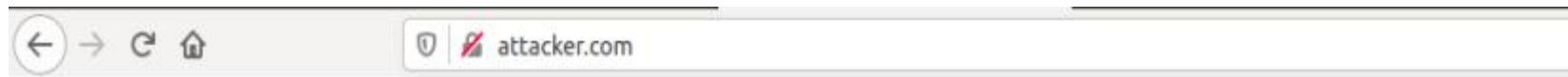


1. В конфиге добавляем строки для проставления кук

```
server_name attacker.com;
location / {
    # First attempt to serve request as file, then
    # as directory, then fall back to displaying a 404.
    add_header "Set-Cookie" "test1=attacker-to-attacker; Max-age=3600; Domain=attacker.com";
    add_header "Set-Cookie" "test2=attacker-to-victim; Max-Age=3600; Domain=victim.com";
}
```

Кука появилась в хранилище домена attacker.com



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

Инспектор Консоль Отладчик Сеть Стили Профайлер Память Хранилище Поддержка доступности					
Indexed DB	Поиск элементов				
Куки	Имя	Значение	Domain	Path	Срок действия / Макс. возраст
http://attacker.com	test1	attacker-to-attacker	attacker.com	/	Fri, 10 Apr 2020 15:39:25 GMT

Проверим на victim.com

The screenshot shows a web browser window with the address bar displaying 'victim.com'. The main content area displays the 'Welcome to nginx!' message, stating that the nginx web server is successfully installed and working, and that further configuration is required. It also provides links to online documentation and support at nginx.org and commercial support at nginx.com. A thank you message 'Thank you for using nginx.' is also present.

Below the browser window, the Chrome DevTools 'Хранилище' (Storage) panel is open. The left sidebar shows a tree view with 'Indexed DB' and 'Куки' (Cookies) expanded. Under 'Куки', there is a single entry for 'http://victim.com'. The right pane of the Storage panel displays the message 'Для выделенного хоста нет данных' (No data for the selected host).

Кука не проставилась, что ожидаемо, т.к. эти домены из разных ориджинов

2. Дана страничка

```
<body>
<script>
  function changeBodyColor(color) {
    document.body.style.backgroundColor = color;
  }
</script>
<button onclick="changeBodyColor('red')">Make it hell!</button>
<button onclick="changeBodyColor('green')">Make it grass!</button>
</body>
```

Внесем дополнения, чтобы цвет фона хранился локальном хранилище и при загрузке страницы загружался последний цвет

```
<body>
  <script>
    function changeBodyColor(color) {
      document.body.style.backgroundColor = color;
      localStorage.setItem('bodyColor', color);
    }
    changeBodyColor(localStorage.getItem('bodyColor'));
  </script>
  <button onclick="changeBodyColor('red')">Make it hell!</button>
  <button onclick="changeBodyColor('green')">Make it grass!</button>
</body>
```

Проверяем, цвет заносится в локальное хранилище

The image displays two screenshots of a web browser's developer tools, specifically the 'Хранилище' (Storage) tab, demonstrating how the 'bodyColor' property is stored in the local storage.

Top Screenshot:

- The browser address bar shows `localhost/test.html`.
- The page background is green.
- Buttons on the page are 'Make it hell!' and 'Make it grass!'.
- The 'Хранилище' (Storage) tab is active, showing a table with the following data:

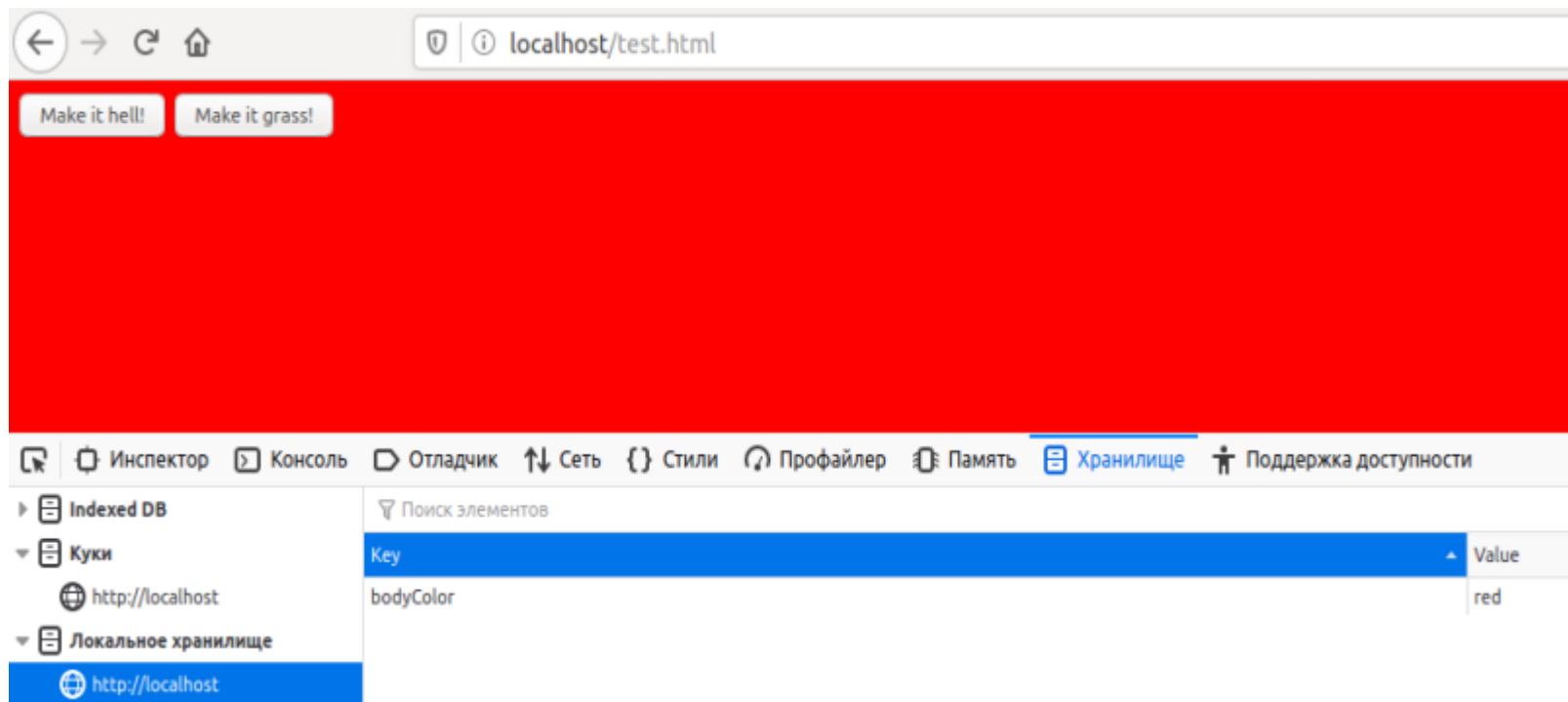
Key	Value
bodyColor	green

Bottom Screenshot:

- The browser address bar shows `localhost/test.html`.
- The page background is red.
- Buttons on the page are 'Make it hell!' and 'Make it grass!'.
- The 'Хранилище' (Storage) tab is active, showing a table with the following data:

Key	Value
bodyColor	red

Закроем браузер и откроем снова, зайдём на страницу, фон восстановился из локального хранилища

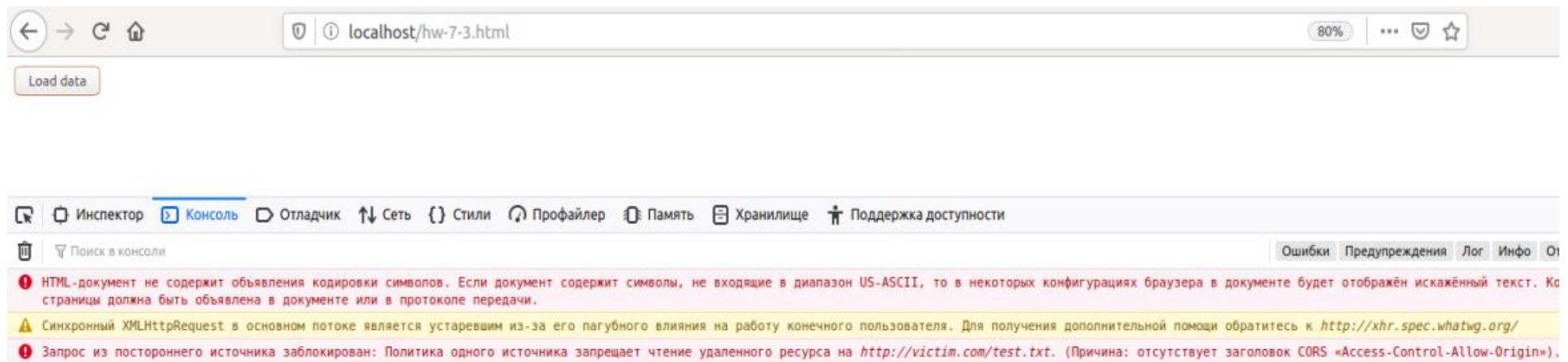


3. Создаем файл, отправляющий кросдоменный XHR запрос

```
<script>
function xhrTest(){
  var xhr = new XMLHttpRequest();
  xhr.open("GET", "http://victim.com/test.txt", false);
  xhr.send();

  if (xhr.status != 200) {
    alert(xhr.status + ': ' + xhr.statusText);
  } else {
    alert(xhr.responseText);
  }
}
</script>
<button onclick="xhrTest()">Load data</button>
```

При запросе с localhost данные не поступают, т.к. отсутствует заголовок CORS, разрешающий чтение другому оиджину



Если добавить заголовок CORS

```
server_name victim.com;

location / {
    # First attempt to serve request as file, then
    # as directory, then fall back to displaying a 404.
    add_header 'Access-Control-Allow-Origin' 'http://localhost';
}
```

То браузер позволяет прочитать данные

The screenshot shows a web browser window with the address bar displaying `localhost/hw-7-3.html`. The page content includes a "Load data" button, a dialog box with the text "Some text here" and an "OK" button. Below the browser window, the Chrome DevTools Network tab is open, showing a list of requests. The third request, to `test.txt` on `victim.com`, is selected and highlighted in blue. The right-hand pane shows the details of this request, including the status (200 OK) and the response headers.

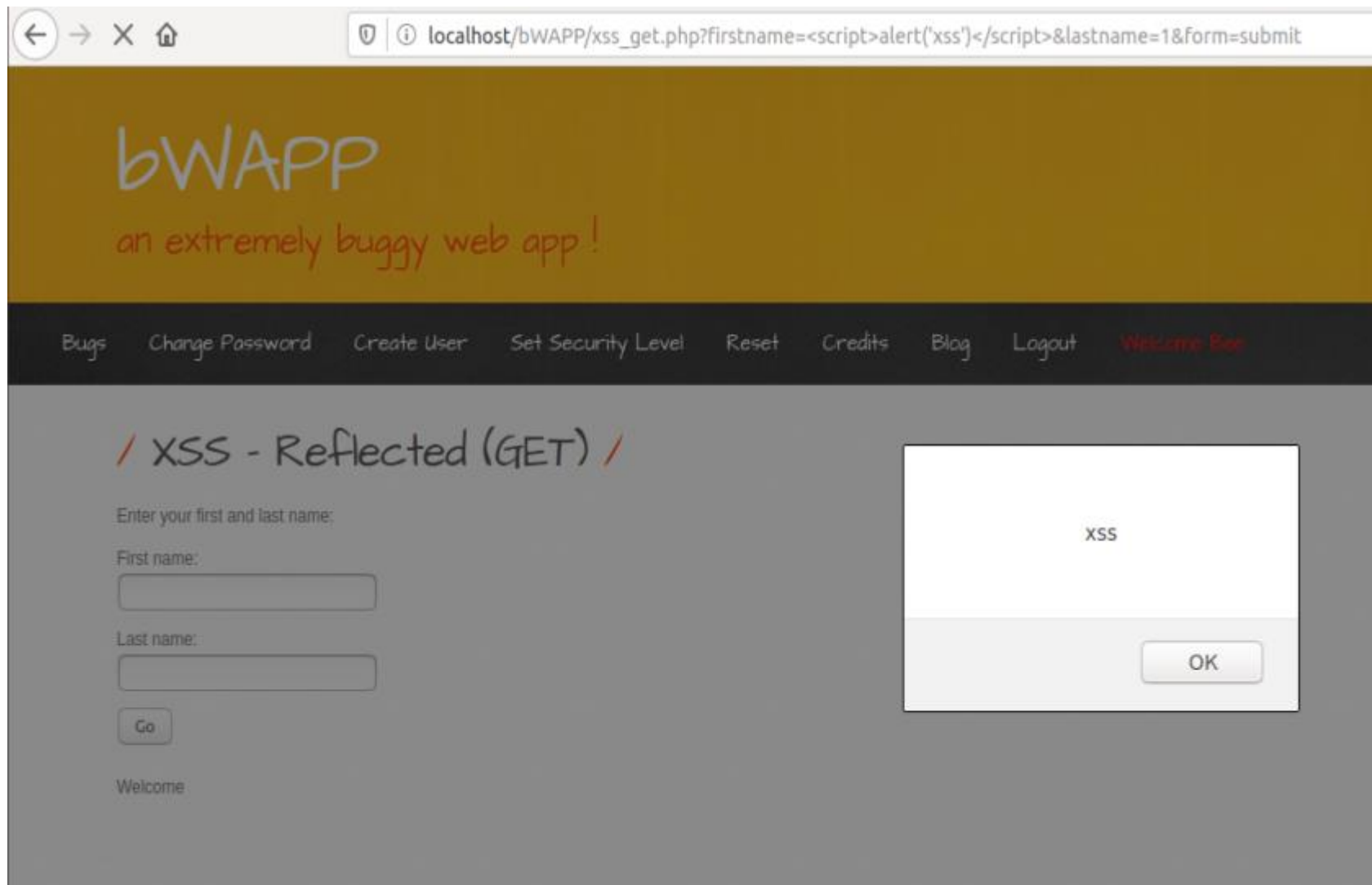
Статус	Метод	Домен	Файл	Причина	Тип	Передано	Размер
200	GET	localhost	hw-7-3.html	document	html	487 Б	315 Б
404	GET	localhost	favicon.ico	img	html	336 Б	178 Б
200	GET	victim.com	test.txt	xhr	plain	307 Б	15 Б

Заголовки ответа (292 Б)

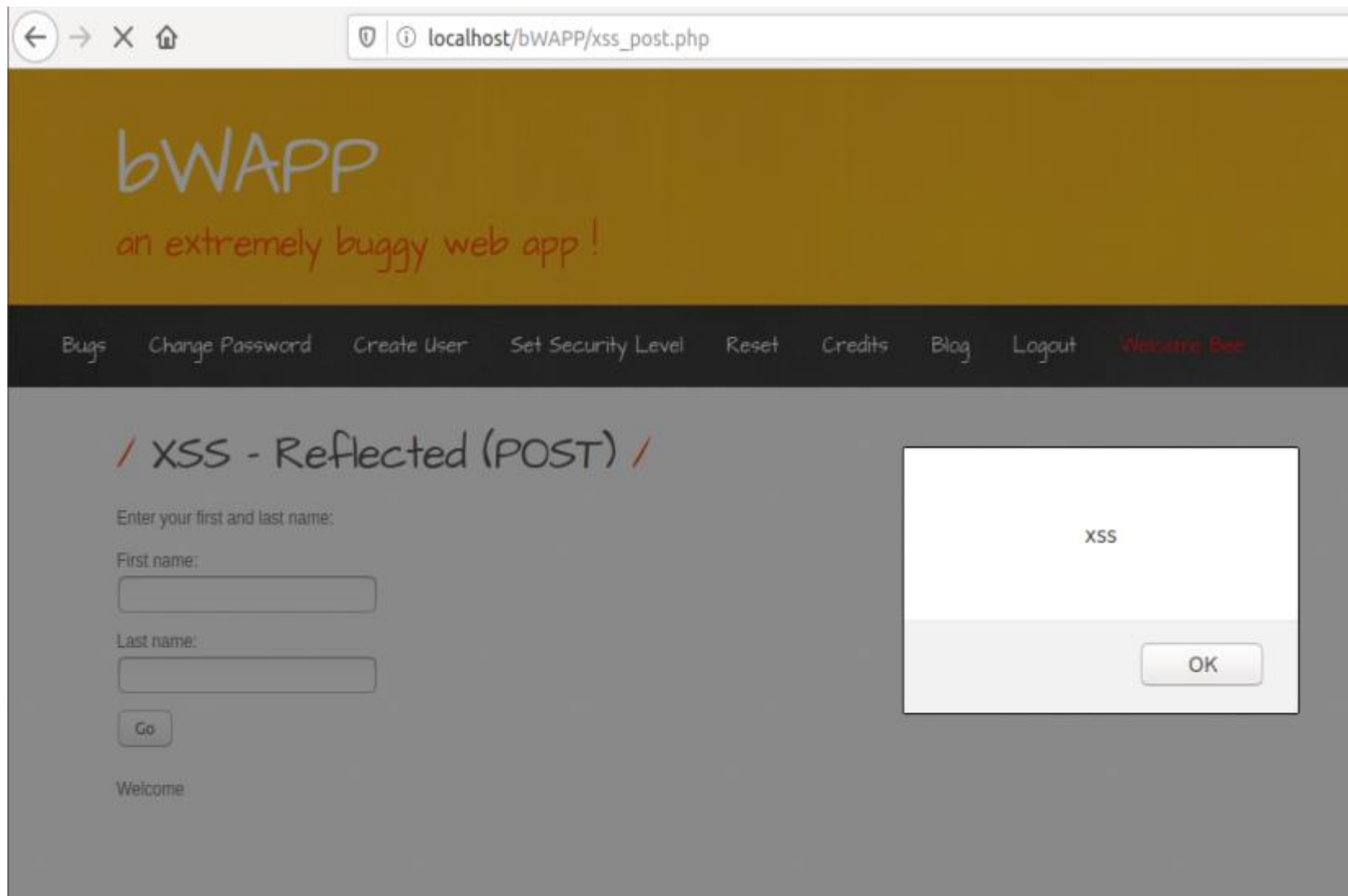
- Accept-Ranges: bytes
- Access-Control-Allow-Origin: http://localhost
- Connection: keep-alive
- Content-Length: 15
- Content-Type: text/plain
- Date: Fri, 10 Apr 2020 19:22:06 GMT
- ETag: "Se7e1bfa-f"
- Last-Modified: Fri, 27 Mar 2020 15:30:02 GMT
- Server: nginx/1.14.0 (Ubuntu)

4. Ищем XSS на уровне low

XSS GET – просто вставляем вредоносный скрипт в URL или в форму



XSS POST – то же самое, только с URL не работает. Форма по прежнему уязвима



XSS JSON

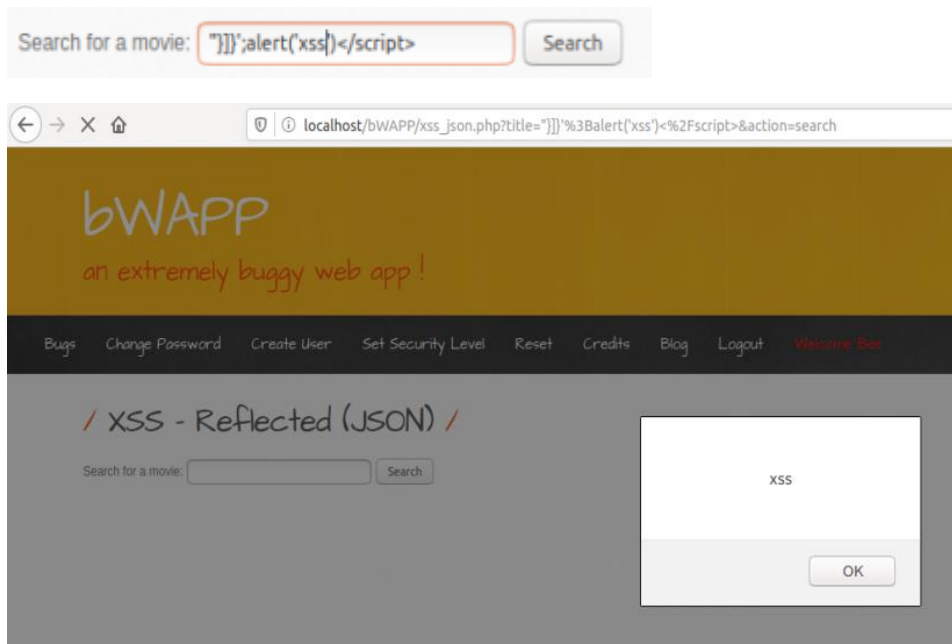
Первая подсказка

```
??? Sorry, we don't have that movie :(')'; // var JSONResponse = eval("(" + JSONResponseString + ")"); var  
JSONResponse = JSON.parse(JSONResponseString);  
document.getElementById("result").innerHTML=JSONResponse.movies[0].response;
```

Вторая: вводим в поиск какую-то строку (например, xss) смотрим ответ сервера

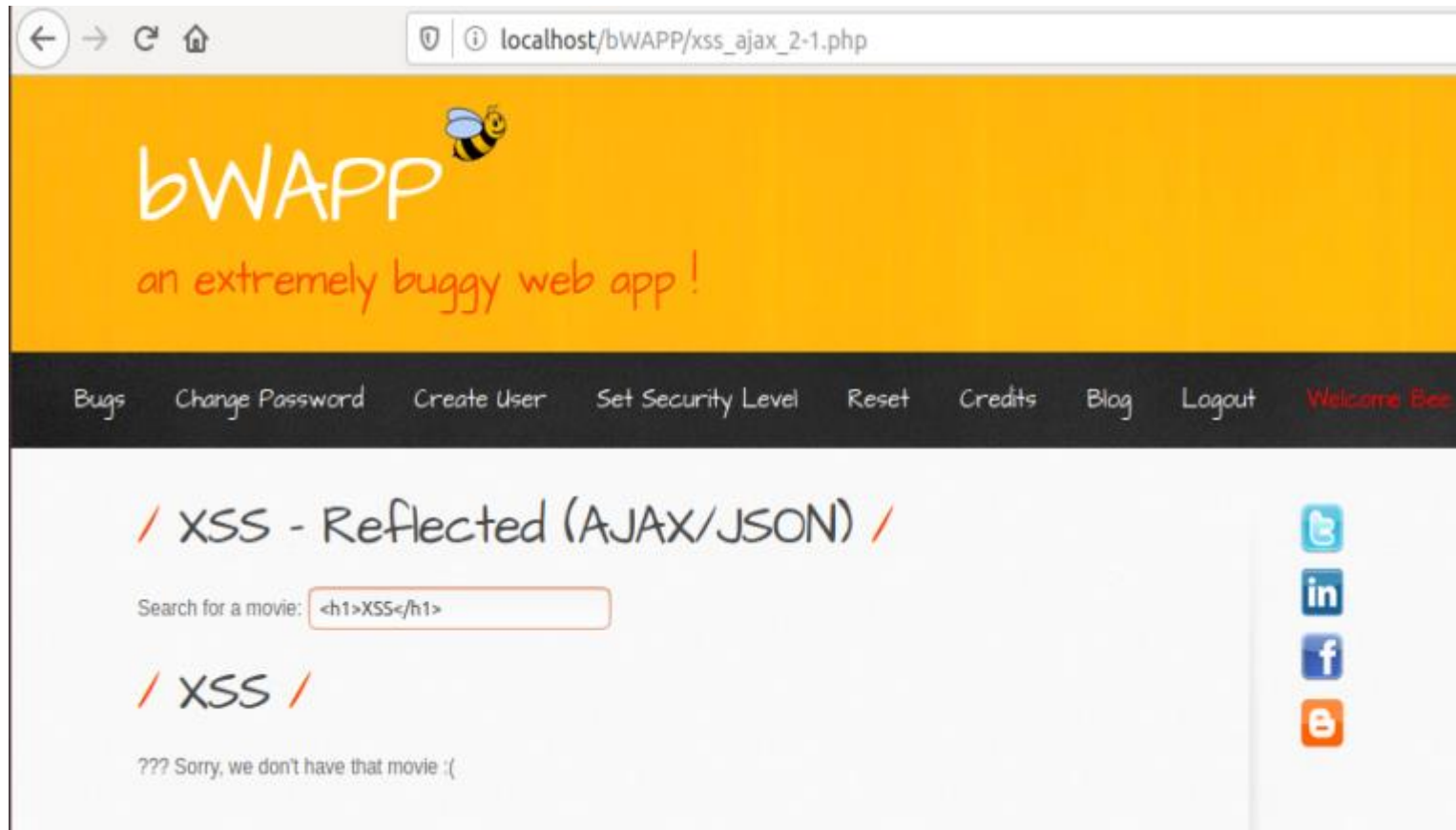
```
<script>  
var JSONResponseString = '{"movies":[{"response":"xss??? Sorry, we don't have that movie :(')'}]';  
// var JSONResponse = eval("(" + JSONResponseString + ")");  
var JSONResponse = JSON.parse(JSONResponseString);  
document.getElementById("result").innerHTML=JSONResponse.movies[0].response;  
</script>
```

Значит если ввести в строку поиска символы “}]}’; то JSON в этом месте закончится, далее пишем вредоносный скрипт и закрываем тэг, например:

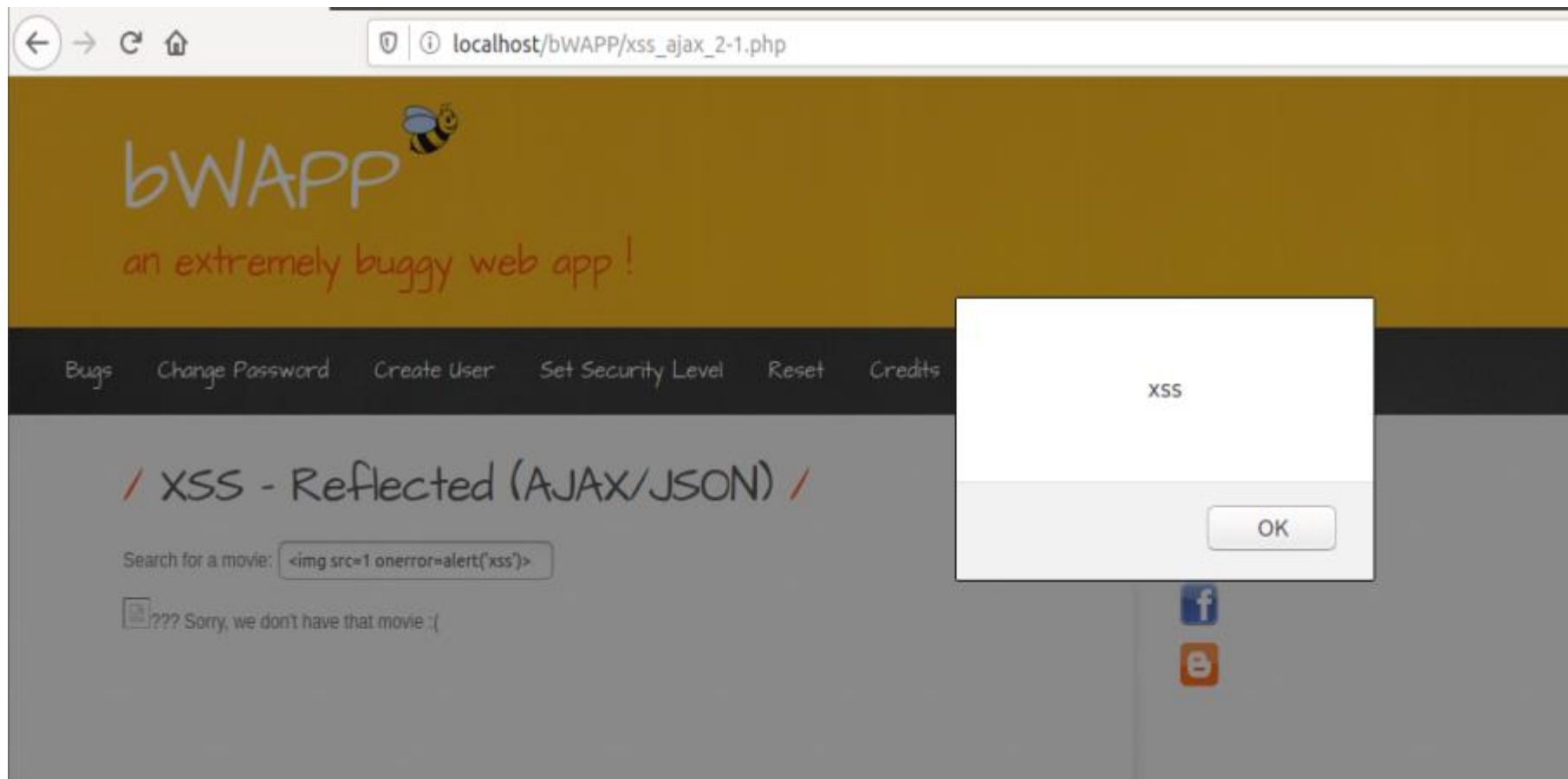


XSS AJAX/JSON (тут пришлось подсмотреть, сам не догадался)

Если вставить тэг, то текст ответа редактируется



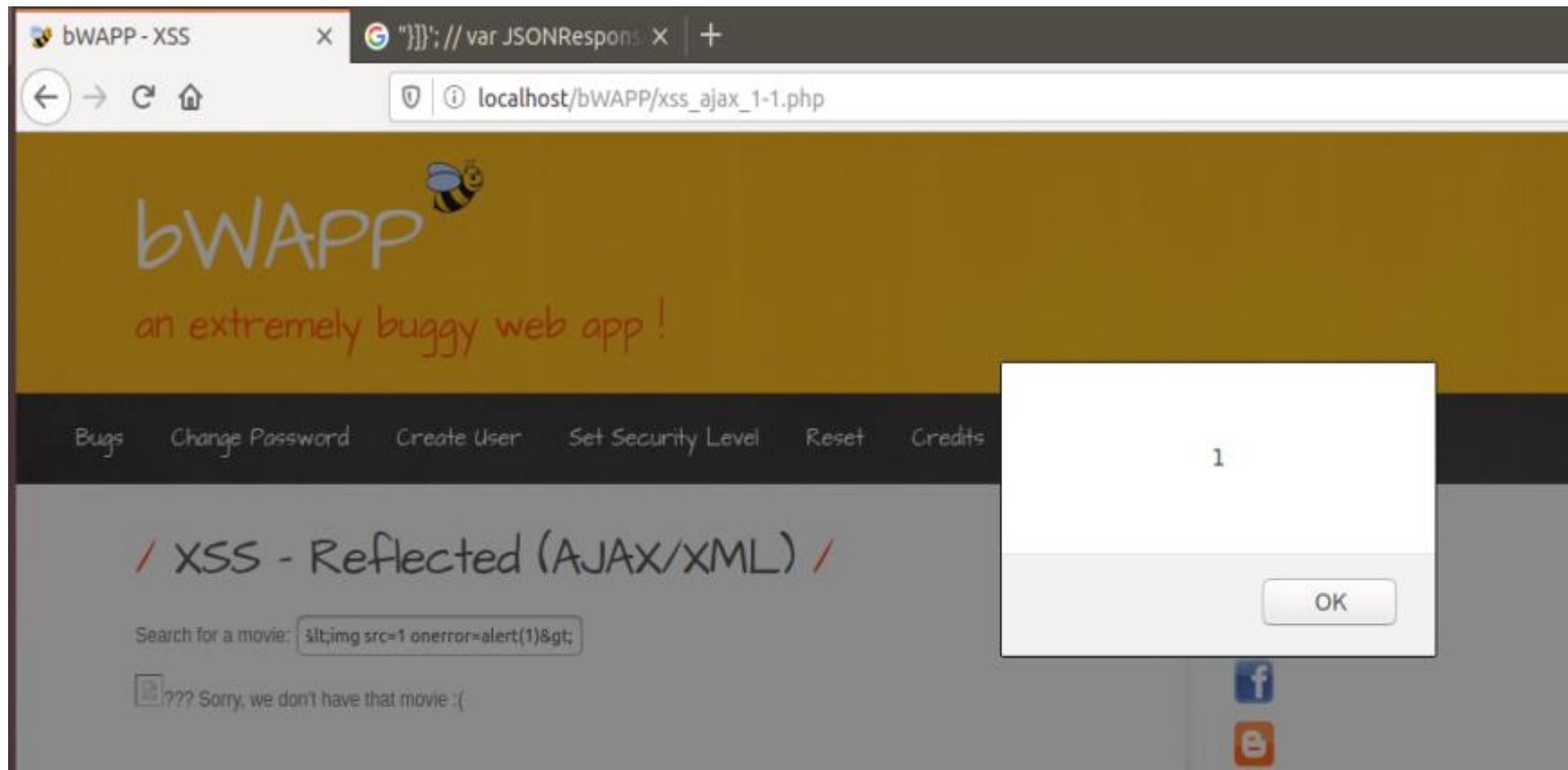
Вставим тэг вызывающий ошибку, такой картинки нет на сайте



XSS AJAX/XML (тоже пришлось подсмотреть)

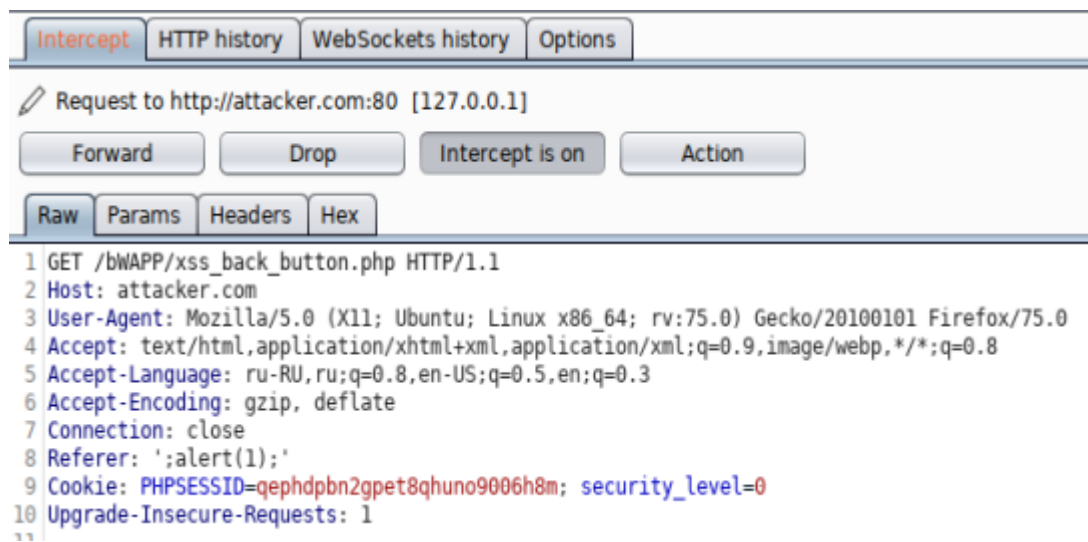
Как в примере выше, только символы тэгов в экранированном представлении

``

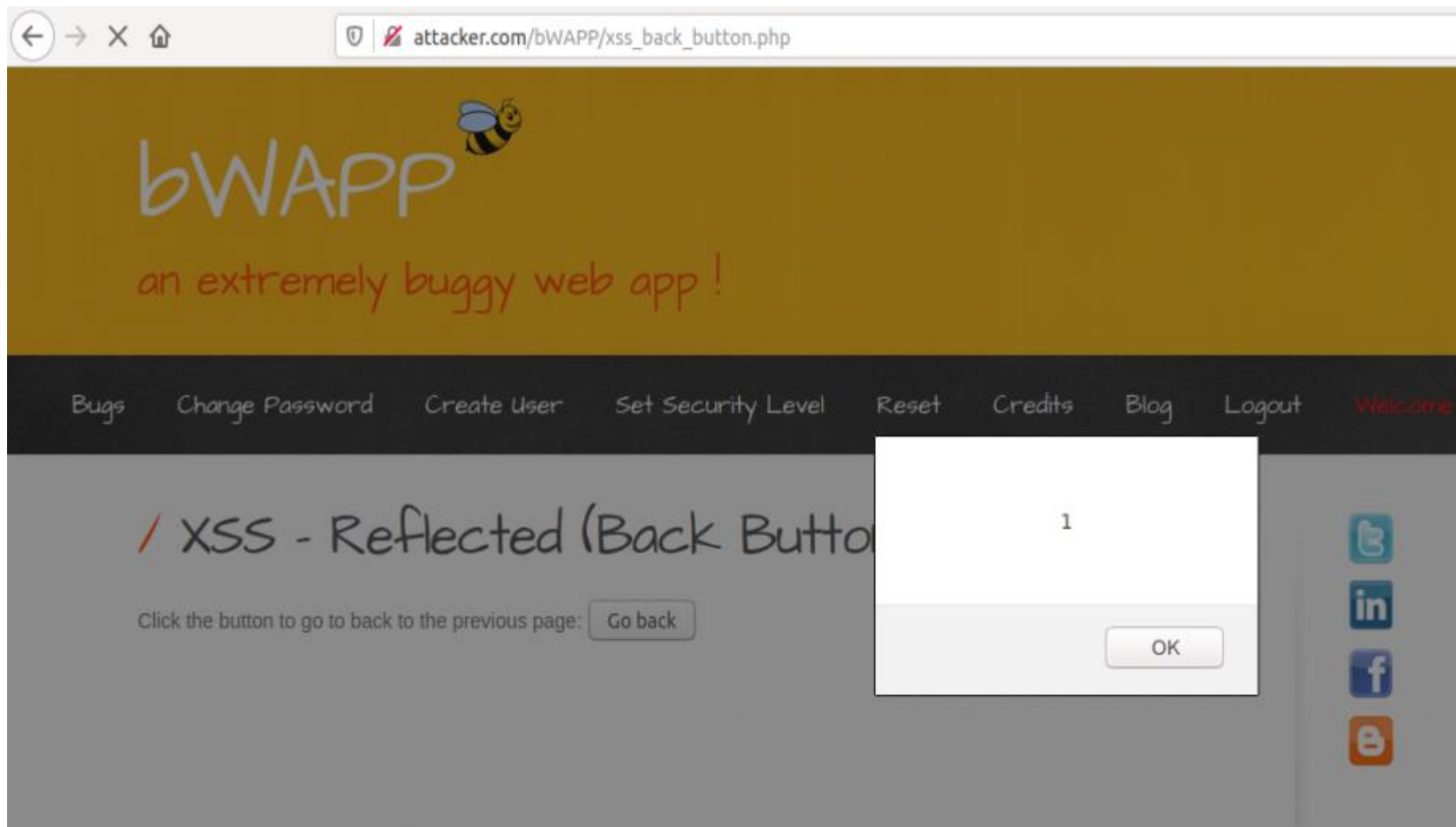


XSS Back button (аналогично не справился)

Смысл в том, что мы перехватываем нажатие на кнопку Burp Suit'ом заменяем referer на ';alert(1);'



Теперь после нажатия на кнопку появляется алерт




XSS Custom Header

Intercept

HTTP history

WebSockets history

Options

 Request to http://attacker.com:80 [127.0.0.1]

Forward

Drop

Intercept is on

Action

Raw

Params

Headers

Hex

1 GET /bWAPP/xss_custom_header.php HTTP/1.1

2 Host: attacker.com

3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:75.0) Gecko/20100101 Firefox/75.0

4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

5 Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3

6 Accept-Encoding: gzip, deflate

7 Referer: http://attacker.com/bWAPP/portal.php

8 bWAPP: <script>alert(1)</script>

9 Connection: close

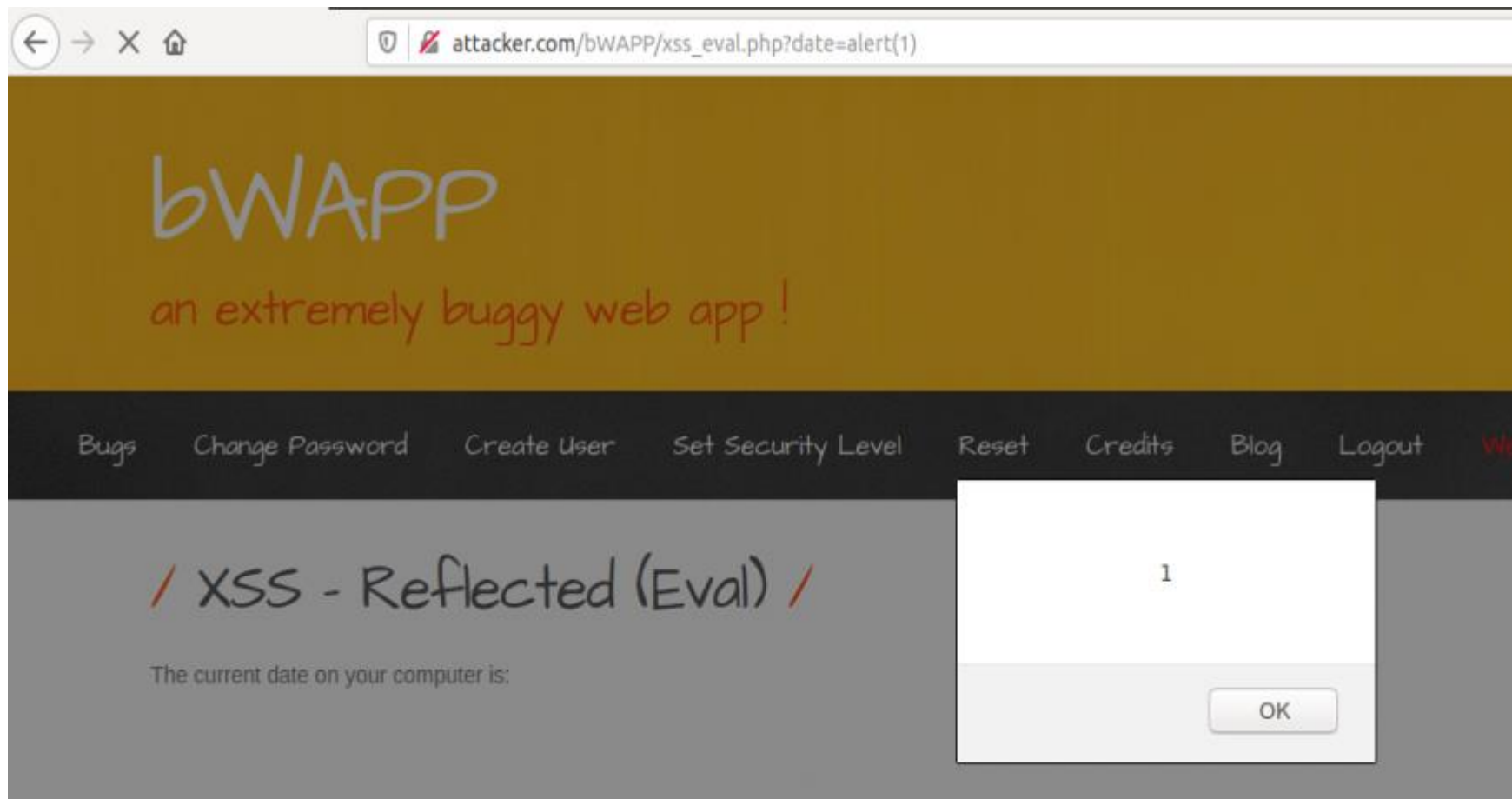
10 Cookie: security_level=0; PHPSESSID=fg2a30dd66neidjlqfc8glrg67

11 Upgrade-Insecure-Requests: 1

12

XSS eval

Вместо даты пишем алерт



XSS PHP_self

Вставляем скрипт в поле формы

